View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

ELSEVIER
Theoretical Computer Science 295 (2003) 141–151
Theoretical Computer Science
www.elsevier.com/locate/tcs

On the computational complexity of infinite words ☆

Pavol Ďuriš[a], Ján Maňuch[b,*]

[a] *Department of Computer Science I (Algorithms and Complexity), University of Technology, Ahornstrasse 55, Aachen, Germany*
[b] *Turku Ctr. for Computer Science and Department of Mathematics, University of Turku, Datacity, Lemminkäisenkatu 14A, Turku, Finland*

Received 21 August 2001; received in revised form 20 February 2002; accepted 20 March 2002

Abstract

This paper contains answers to several problems in the theory of the computational complexity of infinite words. We show that the problem whether all infinite words generated by iterating deterministic generalized sequential machines have logarithmic space complexity is equivalent to the open problem asking whether the unary classes of languages in P and in DLOG are equivalent. Similarly, the problem to find a concrete infinite word which cannot be generated in logarithmic space is equivalent to the problem to find a concrete language which does not belong to DSPACE(n). Finally, we separate classes of infinite words generated by double and triple D0L TAG systems.
© 2002 Elsevier Science B.V. All rights reserved.

MSC: 68Q05; 68Q15; 68Q30

Keywords: Infinite words; Computational complexity; dgsm; Double and Triple D0L TAG systems

0. Introduction

The study of infinite words is an important research topic in combinatorics on words. The main stream of research focused on combinatorial properties, cf. [6], and the descriptional complexity of infinite words, i.e., the measure how complicated simple mechanisms are needed to generate particular infinite words. An extensive study of different machineries for infinite word generation can be found in [1].

☆ Research supported by DAAD and Academy of Finland under a common grant No. 864524.
* Corresponding author.
E-mail addresses: duris@dcs.fmph.uniba.sk (P. Ďuriš), manuch@cs.utu.fi (J. Maňuch).

0304-3975/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0304-3975(02)00400-0

In [3] a new area of investigation was introduced, the computational complexity of words, i.e., the measure how much resources (such as time and space) are needed to generate a certain infinite word. The paper concentrates on relations between the two mentioned complexities: descriptional and computational. Further results in this area can be found in [2].

In [1–3] several interesting problems are proposed. In this paper we will show that even some of the simplest problems proposed are equivalent to well-known hard open problems in the complexity theory of Turing machines.

The paper is organized as follows:

In Section 1 we fix our terminology. In Section 1.1 we recall the definition of the computational complexity, while in Section 1.2 we define several simple methods for generating infinite words:

- iterating a morphism, the most commonly used method introduced already in [7];
- iterating a deterministic generalized sequential machine (a dgsm), i.e. a deterministic finite state transducer;
- double and triple D0L TAG systems.

In Section 2.1 we consider the open problem, proposed in [3], namely whether all infinite words generated by iterating dgsms have logarithmic space complexity. This problem was attacked in [5], claiming that the answer to the problem is affirmative. On the other hand, here we show that the problem is equivalent to an other hard open problem asking whether unary classes of languages P and DLOG (denoted u-P and u-DLOG, respectively) are equivalent. One can easily observe that u-P = u-DLOG if and only if $\bigcup_{c>0} \mathrm{DTIME}(c^n) = \mathrm{DSPACE}(n)$.

Section 2.1 contains two other small results concerning the growth of dgsms. It was shown in [3] that the greatest possible growth of a dgsm is exponential and that infinite words generated by such dgsms have logarithmic space complexity. We show that the smallest non-trivial growth is $\Theta(n \log n)$ and that, similarly, dgsms with such a growth generate infinite words with logarithmic space complexity.

In [3] another interesting problem is proposed: to find a concrete infinite word which cannot be generated in logarithmic space. It is mentioned already in [2] that this problem is at least as hard as to prove $L \notin \mathrm{DLOG}$ for some $L \in \mathrm{NP}$. In Section 2.2 we show that it is exactly as hard as the problem to find a concrete language, which does not belong to DSPACE(n). Note that even the problem to find a concrete language, which does not belong to $\mathrm{DSPACE}(\log n) = \mathrm{DLOG}$ is a hard open problem.

Finally, in Section 2.3 we separate the classes of infinite words generated by double and triple D0L TAG systems as it was conjectured in [1].

1. Preliminaries

In this section we fix our terminology. Let Σ be a finite alphabet. The sets of all finite and infinite words over Σ are denoted Σ^* and $\Sigma^{\mathbb{N}}$, respectively. In the following section we define the computational complexity of infinite words and describe several iterative devices generating infinite words.

1.1. The computational complexity of infinite words

The best way how to define the computational complexity of an object is to describe it in the terms of Turing machines. For example, the Kolmogorov complexity of a finite word is the size of the smallest Turing machine generating the word, cf. [4]. In the case of infinite words we will use the model of computation based on the *k-tape Turing machine*, which consists of

(1) a finite state control;
(2) k one-way infinite working tapes (we assume that there is a beginning on the left of the tape, but the tape is infinite to the right) each containing one two-way read/write head (i.e., the head can move in both directions within the tape);
(3) one infinite output tape containing one one-way write-only head.

We assume that the k-tape Turing machine starts in the initial state with all tapes empty and behaves as an usual Turing machine. We say that the k-tape Turing machine generates an infinite word $w \in \Sigma^{\mathbb{N}}$ if

(1) in each step of the computation, the content of the output tape is a prefix of w;
(2) for each prefix u of w, there is an integer n such that u is a prefix of the content of the output tape after n steps of the computation.

Let M be a k-tape Turing machine generating a word w. The *time* and *space complexities* of M are functions $T_M : \mathbb{N} \to \mathbb{N}$ and $S_M : \mathbb{N} \to \mathbb{N}$ defined as follows:

- $T_M(n)$ is the smallest number of steps of the computation of M when the prefix of w of length n is already written on the output tape;
- $S_M(n)$ is the space complexity of working tapes during first $T_M(n)$ steps of the computation, i.e., the maximum of lengths of words written on working tapes.

Finally, for any integer function $s : \mathbb{N} \to \mathbb{N}$ we define the following complexity classes:

- $\mathrm{GTIME}(s) = \{w \in \Sigma^{\mathbb{N}}$; there exists a k-tape Turing machine M generating w and $T_M(n) \leqslant s(n)$ for all $n \geqslant 1\}$;
- $\mathrm{GSPACE}(s) = \{w \in \Sigma^{\mathbb{N}}$; there exists a k-tape Turing machine M generating w and $S_M(n) \leqslant s(n)$ for all $n \geqslant 1\}$;

It follows from the speed-up argument, as in ordinary complexity theory, that functions $s(n)$ and $cs(n)$, where c is a constant, have the same space complexity classes of infinite words.

1.2. Iterative devices generating infinite words

In this section we define several simple methods used for generating infinite words. The simplest and most commonly used method is to iterate a morphism $h : \Sigma^* \to \Sigma^*$: if h is non-erasing and for a letter $a \in \Sigma$, a is a prefix of $h(a)$, then there exists the limit

$$w = \lim_{n \to \infty} h^n(a).$$

A natural generalization of this method is to use a more powerful mapping in the iteration: a *deterministic generalized sequential machine*, a *dgsm* for short.

A dgsm τ is defined by
(1) a finite set of states Q;
(2) the initial state $q_i \in Q$;
(3) an input alphabet Σ and an output alphabet Δ (we will assume $\Delta = \Sigma$ in this note to be able to iterate the dgsm τ);
(4) a transition relation $\delta \subseteq Q \times \Sigma \times \Delta^* \times Q$, where δ is a partial function $Q \times \Sigma \to \Delta^* \times Q$

A sequence of transitions

$$\alpha = (q_i, u_1, v_1, q_1)(q_1, u_2, v_2, q_2)\ldots(q_{k-1}, u_k, v_k, q_k)$$

is a computation of τ with the input $I(\alpha) = u_1 u_2 \ldots u_k$ and the output $O(\alpha) = v_1 v_2 \ldots v_k$. Obviously, for an input $u \in \Sigma^*$ there exists at most one computation α of τ such that $I(\alpha) = u$. Hence, the mapping $\tau(u) = O(I^{-1}(u))$ is a well-defined partial function. As a convention we assume throughout that all dgsms are non-erasing, i.e., $\delta \subseteq Q \times \Sigma \times \Delta^+ \times Q$.

The further generalization of above methods leads to *double D0L TAG systems* which consist of two infinite one-way tapes each containing a one-way read-only head and a one-way write-only head. In each step of the generation both read-only heads read a symbol and move right to the next square while the write-only heads write the corresponding outputs to the first empty squares of these tapes. We assume that the infinite word generated by a double D0L TAG system is written on the first tape. A double D0L TAG system can be specified in the terms of rewriting rules of the form:

$$\begin{pmatrix} a \\ b \end{pmatrix} \to \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \quad \text{where } a, b \in \Sigma,\ \ \alpha, \beta \in \Sigma^+.$$

Assuming that in each rewriting rule, $|\beta| = 1$, we get a mechanism which iterates a dgsm. Finally, we can define *triple D0L TAG systems* by extending the number of tapes to three.

2. The results

2.1. Dgsms

In [3] the following problem is proposed: are all words generated by dgsms in GSPACE($\log n$)? We prove here that this problem is equivalent to the problem whether classes u-DLOG and u-P are equivalent. Note that this in some sense contradicts the result of [5] that all infinite words generated by iterating dgsms have logarithmic space complexity: if the result in [5] is correct, then, together with the following theorem, we have D-EXPTIME = DSPACE(n), which is unlikely. Lepistö [5] gives only a sketch of the proof of the result and, hence, we are unable to check if it is correct.

Theorem 1. *All infinite words generated by iterating dgsms have logarithmic space complexity if and only if* u-P = u-DLOG.

Proof. First, let us assume that u-P = u-DLOG. Take a dgsm τ over a finite alphabet Σ generating an infinite word $w = w_1 w_2 \dots$. We prove that the space complexity of w is $\mathcal{O}(\log n)$. It is obvious that there is a 1-tape Turing machine M generating the word w in quadratic time. Consider the languages $L_c = \{0^n;\ n \geqslant 1,\ w_n = c\}$ for all $c \in \Sigma$. Note that L_c is an unary language. We can easily construct a Turing machine recognizing L_c in quadratic time using the Turing machine M. By the assumption there exist Turing machines M_c recognizing the languages L_c in logarithmic space. Now, consider a 3-tape Turing machine, which runs M_c's to generate the nth letter of w by using the third tape as a working tape. It stores the binary representation of n on the first tape and the position of the head of Turing machine M_c on the second tape. Before each run of any M_c it erases the working tape and writes "the position 1" on the second tape. It runs M_c for each letter c of the alphabet of τ until some M_c accepts and then it writes the letter c on the output tape. In each step of the simulation of any M_c it checks whether the position represented on the second tape is the last one. Clearly such a machine generates the word w in logarithmic space.

Second, assume that all words generated by iterating a dgsm have logarithmic space complexity. Take a Turing machine M working in polynomial time, i.e., $T(M) = \mathcal{O}(n^k)$, recognizing a language $L \subseteq 0^*$. We construct a 1-tape Turing machine M' with the tape divided into three layers. On the first layer it generates unary inputs in increasing order, on the second layer it simulates computations of M on the input stored on the first layer, and on the third layer it writes 1, if the computation ends in an accepting state, or 0, if it ends in a rejecting state. Before each simulation it erases the second and third layers of the tape.

Now, consider a dgsm τ which carries out the computations

$$C_i \to C_{i+1} \quad \text{for } i \geqslant 0,$$

where C_i corresponds to the ith configuration of M'. It also maps the starting letter \$ into the starting configuration of M'. Clearly, the iteration of τ will generate an infinite word: the sequence $W = \$C_0 C_1 C_2 \dots$ of all configurations of the computation of the Turing machine M'. By the assumption the infinite word W has logarithmic space complexity, i.e., there exists a Turing machine M'' generating W in logarithmic space. Finally, we define a Turing machine M''' recognizing L, which on the input 0^n runs M'', but instead of writing the bits of W to the output tape, it compares its input with the input on the first layer of each generated configuration, and moreover, it checks the first letter on the third layer of each generated configuration. When the compared inputs coincide and the first letter on the third layer is 0 or 1 then the Turing machine M''' halts in the rejecting or in the accepting state, respectively. Otherwise, it continues in generating bits of the next configuration. Clearly, M''' recognizes the language L.

Now, it suffices to show that M''' works in logarithmic space. Let C_{i_n} be the configuration in which M' writes 0 or 1 on the first place of the third layer, while the first layer contains 0^n. Hence in the block of configurations $B_n = C_{i_{n-1}+1} \dots C_{i_n}$, M' erases the second and the third layer of the tape, changes the input on the first layer to 0^n and runs the Turing machine M on this input. Since M works in time $\mathcal{O}(n^k)$ the length of any configuration in the block B_n is at most $\mathcal{O}(n^k)$ and the number of configurations in B_n is at most $\mathcal{O}(n^k)$. Hence the length of the block B_n is at most $\mathcal{O}(n^{2k})$. The Turing

machine M''' generates the first x blocks of configurations on the input 0^x until it halts. Hence it generates the prefix of the infinite word W of the length

$$\sum_{n=0}^{x} |B_n| = \mathcal{O}(x^{2k+1}).$$

Since M'' works in logarithmic space, M''' will use space $\mathcal{O}(\log x)$ to carry out the computation on the input 0^x. □

Next, we will show two small results concerning the growth of a dgsm τ, i.e., the integer function $g:\mathbb{N}\to\mathbb{N}$, where $g(n)$ is the length of $\tau^n(a)$. In [3] it was proved that an infinite word generated by a dgsm with the exponential growth has logarithmic space complexity. Note that the exponential growth is the greatest possible growth. Next, we show that the smallest non-trivial growth of a dgsm is $\Theta(n\log n)$ and also that dgsms with the growth $\Theta(n\log n)$ generate infinite words with logarithmic space complexity. More precisely:

Lemma 2. *If a dgsm has the growth* $\mathrm{o}(n\log n)$, *then it generates an ultimately periodic infinite word. Such word can be generated in constant space.*

Proof. Let τ be a dgsm with the growth $\mathrm{o}(n\log n)$ generating an infinite word $w=w_1w_2,\ldots$, with $w_i\in\Sigma$. Let $\tau_q(z)$ (resp. $\sigma_q(z)$) be the output (resp. the last state) of the dgsm τ after reading the input z and starting in the state q.

We define two sequences of words and a sequence of states of the dgsm τ. Let a be the starting symbol, q_0 be the initial state and let $\tau(a)=\tau_{q_0}(a)=av$. Then put

$$u_1 = a, \quad v_1 = v, \quad q_1 = \sigma_{q_0}(a),$$
$$u_n = u_{n-1}v_{n-1}, \quad v_n = \tau_{q_{n-1}}(v_{n-1}), \quad q_n = \sigma_{q_{n-1}}(v_{n-1}).$$

Observe that $\tau^n(a)=u_{n+1}=u_nv_n=\cdots=u_1v_1v_2\ldots v_n$. This implies $1+\sum_{j=1}^{n}|v_j| = |\tau^n(a)|=\mathrm{o}(n\log n)$. Next, we estimate the length of the increment: $|v_n|$. Since the dgsm τ is non-erasing, we have $|v_i|<|v_n|$ for all $i<n$. This implies

$$n|v_n| \leqslant \sum_{j=n+1}^{2n} |v_j| \leqslant \sum_{j=1}^{2n} |v_j| = \mathrm{o}(2n\log 2n) = \mathrm{o}(n\log n).$$

Hence, $|v_n|=\mathrm{o}(\log n)$ and for any constant c there exists an integer n such that $c^{|v_n|}<n$. If we take $c=|\Sigma|+1$, there must be a repetition among the words $v_1,\ldots,v_n$, say $v_i=v_{i+k}$ for some integers $i, i+k\leqslant n$ and $k>0$. Then $v_j=v_{j+k}$ for all $j\geqslant i$, hence the infinite word $w=u_1v_1v_2v_3\ldots$ is ultimately periodic and it can be then generated in constant space, cf. [3], Lemma 3.1. □

Lemma 3. *An infinite word generated by a dgsm with the growth* $\Theta(n\log n)$ *has logarithmic space complexity.*

Proof. Let τ be a dgsm with the growth $\Theta(n \log n)$ generating an infinite word $w = w_1 w_2 \ldots$. Consider the sequences $\{u_n\}_{n \geqslant 0}$, $\{v_n\}_{n \geqslant 0}$, $\{q_n\}_{n \geqslant 0}$ defined in the previous proof.

We construct a Turing machine M generating w as follows. In the first step it writes u_1 on the output tape, v_1 on the first tape and it sets to the state q_1. In each step $n > 1$, it simulates the dgsm τ on the input written on the first tape starting in the state q_{n-1}. The output of the simulation of τ is written, at the same time, to the output tape and to some temporary tape, so that after the simulation it can be copied back to the first tape. Hence, in the end of the step n the first tape contains the word v_n. The last state of the simulation of τ in the step n is q_n, from which the simulation continues in the next step.

The space needed to generate the mth letter of w is at most $|v_n|$, where n is an integer such that v_n contains the letter w_m, i.e.,

$$\sum_{i=1}^{n-1} |v_i| < m \leqslant \sum_{i=1}^{n} |v_i|.$$

One can show in the same way as in the proof of Lemma 2 that $|v_n| = \mathcal{O}(\log n)$. Moreover, since $n \leqslant 1 + \sum_{i=1}^{n-1} |v_i| \leqslant m$, we have $|v_n| = \mathcal{O}(\log n) = \mathcal{O}(\log m)$. Hence, the Turing machine M works in logarithmic space. □

2.2. Logarithmic space complexity

The second part of Problem 5.2 in [2] asks to find a specific infinite word which cannot be generated in logarithmic space. We show that this problem is as hard as the problem to find a specific language which does not belong to DSPACE(n), and this is a hard open problem.

Notation 1. Denote the nth binary word in lexicographical order by lex(n). Note that for $n \geqslant 1$, $\mathrm{bin}(n) = 1\,\mathrm{lex}(n)$.

Definition 4. Let w be an infinite binary word and $L \subseteq \{0,1\}^*$ an binary language. We say that w *determines* the language L if for every positive integer n, the nth letter of w is 1 if and only if lex(n) belongs to L.

Theorem 5. *Let w be an infinite binary word and L the language determined by the word w. Then the word w is in* GSPACE($\log n$) *if and only if L belongs to* DSPACE(n).

Proof. First, assume that w has logarithmic space complexity. Let M be a Turing machine generating w in logarithmic space. We construct a Turing machine M' recognizing the language L. Let lex(n) be the word on the input tape of M', where n is a positive integer. The length of the input is $\Theta(\log n)$. M' simulates M in the following way: it remembers only the last letter generated by M and counts the number of them on a special working tape. When this number is equal to n, it stops and accepts the input if and only if the last output letter was 1.

Since M uses only $\Theta(\log n)$ space to generate the first n output letters and the same space is needed for counting the number of output letters, M' works in space $\Theta(\log n)$, which is linear to the length of the input. Hence, $L \in \mathrm{DSPACE}(n)$.

Next, assume that $L \in \mathrm{DSPACE}(n)$. So we have a Turing machine M recognizing L in linear space. Let M' be a Turing machine such that it generates words in lexicographical order on the first working tape and runs M on each generated word. Depending on if the word was accepted or rejected it writes 1 or 0 on the output tape. Clearly, the length of the nth word on the first working tape is $\Theta(\log n)$. M works in linear space, hence in space $\Theta(\log n)$. Therefore, M' uses logarithmic space to generate the nth letter: $w \in \mathrm{GSPACE}(\log n)$. □

As a consequence we have that if we are able to show about a specific infinite word that it does not belong to $\mathrm{GSPACE}(\log n)$, then we would have also a specific language which does not belong to $\mathrm{DSPACE}(n)$, and vice versa. Note, that even the problem to show that a specific language does not belong to $\mathrm{DSPACE}(\log n)$ is open.

2.3. Separation of double and triple D0L TAG systems

In Section 6 of [3] is mentioned that the generation of infinite words by double D0L TAG systems is a very powerful mechanism, and that it is not known any concrete example of an infinite word which cannot be generated by this mechanism, although by a diagonalization argument such words clearly exist. In [1, Conjecture 4] is conjectured that there exists an infinite word that can be generated by a triple D0L TAG system, but not by any double D0L TAG system. In what follows we are going to give the whole class of infinite words which cannot be generated by any double D0L TAG system. Combining this result with some results in [1] we can also give an affirmative answer to Conjecture 4 of [1]. First, let us fix some notation.

Notation 2. Let $w = c_1 \dots c_n$ be a word with $c_i \in \Sigma$. Then $\mathrm{symb}(w) = \{c_i,\ 1 \leqslant i \leqslant n\}$ denotes the set of all symbols occurring in the word.

Theorem 6. *Let $s : \mathbb{N} \to \mathbb{N}$ be an integer function such that $s(i) \in 2^{\omega(i)}$, i.e., $s(i)$ grows faster than exponentially. Then the infinite word*

$$w = 10^{s(1)}10^{s(2)}10^{s(3)}\dots$$

cannot be generated by any double D0L TAG system.

Proof. Assume that w can be generated by a double D0L TAG system, and let τ be such a system. Let M be the set of all symbols occurring on the second tape of τ and B the maximal number of symbols written on any tape in one step, i.e.,

$$B = \max\left\{\max(|\alpha|, |\beta|),\ \text{where } \begin{pmatrix} a \\ b \end{pmatrix} \to \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \text{ is a rewriting rule of } \tau\right\}.$$

Let $w_i = 10^{s(i)}$. First, we show that for any constant $k \geqslant 1$, there is a positive integer j such that

$$s(j) + 1 = |w_j| > k|w_1 \dots w_{j-1}1| + 1. \tag{1}$$

Since $s(i) \in 2^{\omega(i)}$, for any number $c > 1$, there is an integer j such that

$$|w_j| > c^j,$$
$$|w_i| \leqslant c^i \quad \text{for all } 1 \leqslant i \leqslant j-1.$$

Then, we have

$$k|w_1 \dots w_{j-1}1| \leqslant k \sum_{i=0}^{j-1} c^i \leqslant k \frac{c^j - 1}{c-1}.$$

Taking $c = k+1$, we get $k|w_1 \dots w_{j-1}1| + 1 \leqslant c^j < |w_j|$.

Consider that we are reading the first 0 of $w_j = 10^{s(j)}$ in the word w. Let u_1 (resp. v_1) be the word written on the first (resp. second) tape between the reading and the writing head. And let, recursively, u_l (resp. v_l) be the word added on the first (resp. second) tape after reading v_{l-1}. Note that for all $l \geqslant 1$, we have $|v_1 \dots v_{l-1}| < |u_1 \dots u_l|$.

We define also a set $M_0 \subseteq M$ as follows. For $x \in M$, let $\binom{0}{x} \to \binom{\alpha}{\beta}$ be a rule of τ. Then, $x \in M_0$ if and only if $\alpha \in 0^+$. Hence, M_0 contains every symbol $x \in M$ such that when reading 0 on the first tape and the symbol x on the second tape, it writes only 0's on the first tape.

Assume that for some $i \geqslant 1$

$$\sum_{l=1}^{i} |u_l| \leqslant s(j). \tag{2}$$

By (2), the words $u_1, \dots, u_i$ contain only 0's, hence when reading the words $v_1, \dots, v_{i-1}$, only 0's are written on the first tape. Since $|v_1 \dots v_{l-1}| < |u_1 \dots u_l|$, we have also that while reading the words $v_1, \dots, v_{i-1}$, only 0's are read from the first tape.

Consider the following oriented graph. The vertices are elements of M. There is an arc $x \to y$, if there is a rule $\binom{0}{x} \to \binom{\alpha}{\beta}$ in τ such that $y \in \mathrm{symb}(\beta)$ (see Notation 2). For $X \subseteq M$, let $\mathrm{clos}(X)$ be the set of all vertices of the graph to which we reach from any vertex of X following the arcs. If $\mathrm{clos}(\mathrm{symb}(v_1)) \subseteq M_0$, then since $u_1 \in 0^+$, we have, by induction, that $u_l \in 0^+$ and $\mathrm{symb}(v_l) \subseteq \mathrm{clos}(\mathrm{symb}(v_1)) \subseteq M_0$ for all $l \geqslant 1$. This is a contradiction since there must be u_l containing 1.

Hence, there must be an oriented path starting in a vertex of $\mathrm{symb}(v_1)$ and ending in a vertex of $M - M_0$ of length at most $|M_0|$. Let i_0 be the length of the shortest of such paths. If we prove that (2) holds for $i = |M| + 1 \geqslant |M_0| + 2 \geqslant i_0 + 2$, then since during reading $v_1, \dots, v_{i_0+1}$ only 0's are read and written on the first tape, we have $\mathrm{symb}(v_1), \dots, \mathrm{symb}(v_{i_0}) \subseteq M_0$ and $\mathrm{symb}(v_{i_0+1}) \cap (M - M_0) \neq \emptyset$. This implies that $1 \in \mathrm{symb}(u_{i_0+2})$ contradicting (2).

Now it suffices to prove that (2) holds for $i = |M| + 1$. After reading one symbol, the system τ can write on any tape at most B symbols. Hence, we have $|u_{l+1}|, |v_{l+1}| \leqslant B|v_l|$

for all $l \geqslant 1$. We estimate the left hand side of (2):

$$\sum_{l=1}^{|M|+1} |u_l| \leqslant |u_1| + \sum_{l=2}^{|M|+1} B^{l-1}|v_1| \leqslant \max(|u_1|, |v_1|) \sum_{l=0}^{|M|} B^l. \tag{3}$$

Notice that $T = \sum_{l=0}^{|M|} B^l$ is a constant for τ.

Next, we estimate $|u_1|$ and $|v_1|$. Consider the situation when the reading heads are on the $(n+1)$th symbols of both tapes. Then on each tape n symbols have already been read and hence at most Bn symbols written. So, there is at most $(B-1)n$ symbols between writing and reading head on each tape. This implies

$$\max(|u_1|, |v_1|) \leqslant (B-1)|w_1 \dots w_{j-1}1|. \tag{4}$$

Taking $k = T(B-1)$, we obtain

$$\sum_{l=1}^{|M|+1} |u_l| \overset{(3)}{\leqslant} T \max(|u_1|, |v_1|) \overset{(4)}{\leqslant} T(B-1)|w_1 \dots w_{j-1}1| \overset{(1)}{\leqslant} s(j)$$

as desired. □

Let us recall one result proved in Examples 11 and 13 of [1].

Lemma 7. *Let $s : \mathbb{N} \to \mathbb{N}$ be an integer function which is computable, i.e., can be computed by a Turing machine. Then there exists an integer function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n) \geqslant s(n)$ for all $n \geqslant 1$ and the word*

$$w = 10^{t(1)}10^{t(2)}10^{t(3)}\dots$$

can be generated by a triple D0L TAG system. Moreover, such a function t can be effectively computed.

The proof of the lemma is based on the following idea. Let M be a Turing machine computing unary strings $1^{s(1)}, 1^{s(2)}, \dots$ and let τ be a dgsm generating the sequence of configurations of the computation of M. We can easily extend the dgsm τ to a triple D0L TAG system by coding all letters generated by τ to 0, except for the last letters of the strings in the sequence $1^{s(1)}, 1^{s(2)}, \dots$, which are coded to 1. Together with our result we have the following corollary.

Corollary 8. *There exists an infinite word which can be generated by a triple D0L TAG system, but not by any double D0L TAG system.*

Hence, the inclusion "double D0L $\subseteq$ triple D0L" is proper, as conjectured in Conjecture 4 in [1].

References

[1] K. Culik II, J. Karhumäki, Iterative devices generating infinite words, Internat. J. Found. Comput. Sci. 5 (1) (1994) 69–97.

[2] J. Hromkovič, J. Karhumäki, Two lower bounds on computational complexity of infinite words, in: G. Paun, A. Salomaa (Eds.), New Trends in Formal Languages, Lecture Notes in Computer Science, Vol. 1218, 1997, pp. 366–376.
[3] J. Hromkovič, J. Karhumäki, A. Lepistö, Comparing descriptional and computational complexity of infinite words, in: J. Karhumäki, H.A. Maurer, G. Rozenberg (Eds.), Results and Trends in Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 812, 1994, pp. 169–182.
[4] A.N. Kolmogorov, Three approaches to the quantitative definition of information, Problems Inform. Transmission 1 (1968) 662–664.
[5] A. Lepistö, On the computational complexity of infinite words, in: J. Dassow, G. Rozenberg, A. Salomaa (Eds.), Developments in Language Theory II, World Scientific, Singapore, 1996, pp. 350–359.
[6] M. Lothaire, Combinatorics on words, in: Encyclopedia of Mathematics and its Applications, Vol. 17, Addison-Wesley, Reading, MA, 1983.
[7] A. Thue, Über unendliche zeichenreihen, Norske Vid. Selsk. Skr., I Mat. Nat. KI., Kristiania 7 (1906) 1–22.